Colofon

RAZzies is een uitgave van de Radio Amateurs Zoetermeer. Bijeenkomsten van de Radio Amateurs Zoetermeer vinden plaats op elke tweede en vierde woensdag van de maanden september - juni om 20:00 uur in buurthuis 't Span, Sullivanlijn 31 Zoetermeer.

Website:

https://www.pi4raz.nl

Redactie:

Frank Waarsenburg PA3CNO pa3cno@pi4raz.nl

Eindredactie:

Robert de Kok PA2RDK pa2rdk@pi4raz.nl

Informatie:

info@pi4raz.nl

Kopij en op- of aanmerkingen kunnen verstuurd worden naar razzies@pi4raz.nl

Nieuwsbrief:

https://www.pi4raz.nl/maillist/subscribe.php

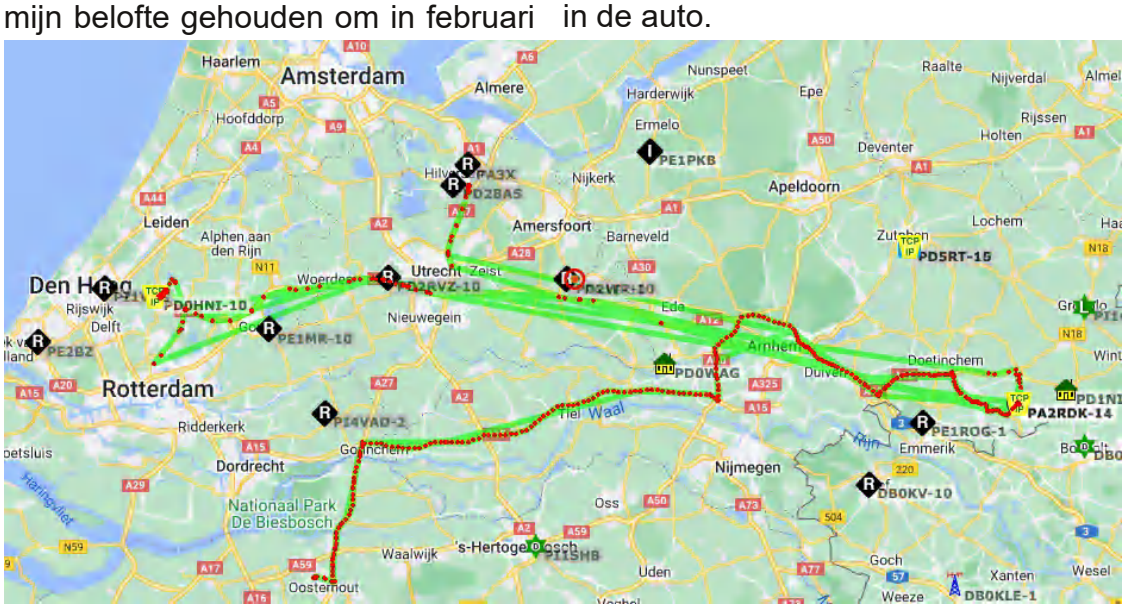
Van de redactie

Het zomert volop buiten. Dat maakt het veel te heet in de shack en soldeerbouten maken dat alleen maar erger. Gelukkig moet ik ook nog wat mechanisch werk doen aan de kast van mijn nieuwe transceiver, maar na het boren van twee gaten heb ik al een natte rug. Ook niet fijn. Dus heb ik de afgelopen tijd niet veel aan de hobby gedaan en wat ik wel gedaan heb zorgde nou niet echt voor veel voortgang in mijn project. Kortom, beter maar wachten tot de temperaturen weer een beetje dragelijk worden zodat prutsen in de shack weer vol te houden is. Maar kennelijk geldt dat niet voor iedereen want de contesten gaan gewoon in alle hevigheid door waardoor er in de weekenden nog steeds geen fatsoenlijke verbinding te maken is. Wat wel opvalt is dat de vakantienetten weer op allerlei frequenties te horen zijn. Dat maakt onze hobby zo leuk: het maakt niet uit waar je zit, maar een amateur ziet altijd wel kans om op een of andere manier contact te houden met de radiovrienden. Het punt is dat we daarin natuurlijk niet de enigen zijn. Het is 's-avonds echt zoeken naar een plekje in de 40m band; de band die bij uitstek geschikt is om binnen Europa verbindingen te maken. Ergens in september zal de rust wel weer weerkeren en pakken ook wij de draad weer op met onze bijeenkomsten en experimenten. Tot die tijd: geniet van de vakantie en laat eens weten welke hobbyspullen er zoal mee gaan.

APRS Transceiver V2

Robert de Kok, PA2RDK

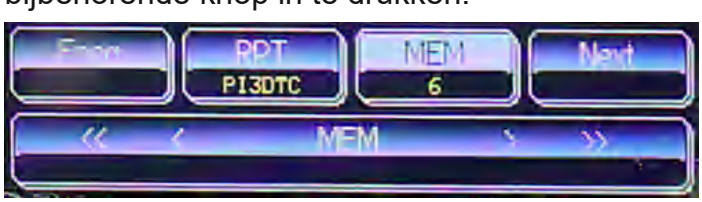
Door allerlei zakelijke beslommeringen, andere projecten en een tripje naar Liechtenstein, heb ik mij niet aan mijn belofte gehouden om in februari een vervolg te schrijven over de APRS transceiver. Deze los ik bij deze in. Onderhand is de transceiver (bijna) klaar en reist die meestal met mij mee in de auto.



Voor in de auto heb ik beugels geprint waardoor de transceiver netjes onder handbereik hangt. Het display vertelt alles wat je weten wilt en is voorzien van alle bedieningsorganen. Alleen het volume is ook met een gewone potmeter te bedienen.

Werkwijze en modi

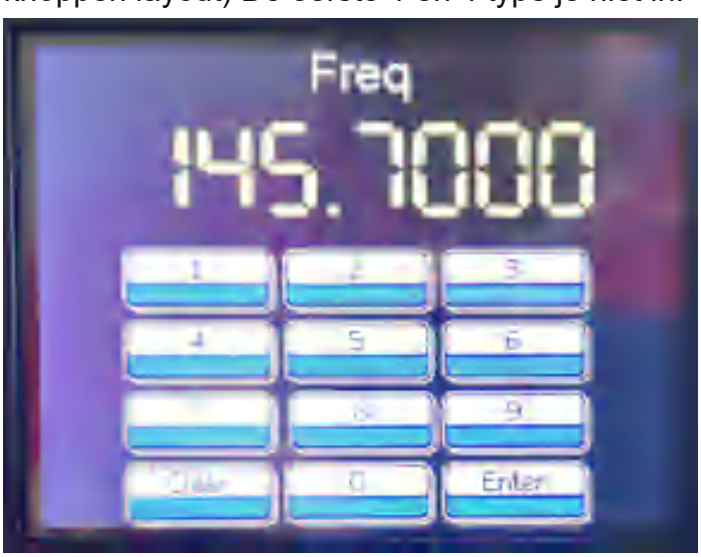
De knoppen zijn verdeeld over 4 pagina's. Met 'prev' en 'next' kan tussen de pagina's worden gewisseld. De laatste pagina is de info pagina. De bedieningsorganen die normaal met een potmeter of rotary encoder worden bediend, worden nu met pijltjes op het scherm bediend. De functie van de pijltjes wordt geregeld door de bijbehorende knop in te drukken.



Druk bijvoorbeeld op MEM. De knop MEM wordt highlighted en in de pijltjesknop komt MEM te staan. Hetzelfde geldt voor de knoppen SQL, Freq, RPT, Vol, Tone (op pagina 2) en Light (op pagina 3). Ook bij Scan werkt het zo en worden de pijltjes gebruikt om de scanmode (stop of auto resume) te selecteren. De Scan knop zelf start en stopt het scannen. Dit scannen is ook te starten door de up of down knop op de microfoon 3 seconden in te drukken. Natuurlijk stopt de PTT-knop op de microfoon het scannen. De transceiver scant alleen omhoog.

De mode Freq(uentie)

De transceiver kent 3 modi: Freq, RPT en MEM. Hierbij loop je met de pijltjes over de band, in stappen van 12.5kHz, standaard van 144 tot 146 MHz. Maar de bovengrens is in de instellingen, waarover later meer, in te stellen. Ook kan de autoshift (vanaf 145.575) worden ingesteld. Als de Freq highlighted is en je drukt nogmaals op Freq, kun je de frequentie ook intypen. (Het plaatje hieronder is met de oude knoppen layout) De eerste 1 en 4 type je niet in.



145.725 wordt dus gekozen door 57250 in te typen. De CTCSS kan met de Tone knop op pagina 2 worden ingesteld. Druk verschillende malen op Tone om te schakelen tussen R(X), T(X) en RT (RX en TX). Met de pijltjes kan de gewenste toon worden geselecteerd. Met de shift knop, ook op pagina 2, kan de gewenste shift (+ of – 600kHz) worden ingesteld.

Als de TX-frequentie door de shift boven de bandgrens uit zou komen, kan de + shift niet worden gekozen. Hetzelfde geldt als de TX-frequentie onder de bandgrens zou uitkomen, dan kan de – shift niet worden geselecteerd. De scanknop scant in de Freq mode de hele band. In de Freq mode werken de pijltjes een beetje anders dan bij de andere functies. Afhankelijk van waar je de knop indrukt, is de stapgrootte 100, 50, 25 of 12.5 kHz.

De mode RPT (repeater)

Hierbij loop je met de pijltjes door de lijst van repeaters. Deze lijst is in de config.h van de software hard geprogrammeerd middels een array van structs.

```
typedef struct {        // Repeaterlist
    const char *name;   // Repeatername
    const char *city;   // Repeatercity
    int8_t shift;       // Shift + or -
    uint16_t channel;   // RX Freq channel (12.5KHz steps, 144.000MHz = 0, channel 128 is 145.600MHz)
    uint16_t ctcssTone; // See list of CTCSS codes in config.h
    uint16_t hasTone;   // 0=disabled, 1=only RX, 2=only TX, 3=RX and TX
} Repeater;
```

```
const Repeater repeaters[] = {
    {"","",0,0,0,0},
    {"PI3UTR","IJsselstein",-1,126,4,2},
    {"PI3ALK","Alkmaar",-1,128,8,3},
    {"PI3ZVL","Overslag",-1,128,0,0},
    {"PI3TWE","Eibergen",-1,128,4,3},
    {"PI3SRT","Venlo",-1,128,2,3},
    {"PI3APM","Appingendam",-1,129,6,3},
    {"PI3BOZ","Bergen/Zoom",-1,130,8,3},
    {"PI3ZOD","Emmen",-1,130,0,0},
    {"PI3AMF","Amersfoort",-1,130,0,0},
    {"PI3FRL","Leeuwarden",-1,131,6,3},
    {"PI3ZUT","Zutphen",-1,131,0,0},
    {"PI3MEP","Meppel",-1,132,6,3},
    {"PI3BRD","Breda",-1,132,2,3},
    {"PI3VLI","Vlissingen",-1,133,8,3},
    {"PI3RTD","Rotterdam",-1,134,8,3},
    {"PI3NOV","'t Harde",-1,134,4,3},
    {"PI3VHP","Vroomshoop",-1,135,4,3},
    {"PI3YMD","IJmuiden",-1,135,8,3},
    {"PI3HVN","Heerenveen",-1,136,6,3},
    {"PI3EHV","Eindhoven",-1,136,2,3},
    {"PI3DTC","Doetinchem",-1,136,4,3},
    {"PI3ZAZ","Wormerveer",-1,137,8,3},
    {"PI3GOE","Goes",-1,138,8,3},
    {"PI3APD","Apeldoorn",-1,138,4,3},
    {"PI3ZLB","Geleen",-1,138,2,3},
    {"PI3FLD","Lelystad",-1,139,4,3},
    {"PI3NYM","Nijmegen",-1,140,4,3},
    {"PI3GRN","Groningen",-1,140,6,3},
    {"PI3RAZ","Zoetermeer",-1,140,8,3},
    {"PI3ALM","Almere",-1,141,0,0},
    {"PI3ASD","Amsterdam",-1,142,8,3}
};
```

Als deze lijst aangepast moet worden, dient dit dus in de code te gebeuren. Bij de geselecteerde repeater wordt automatisch de juiste frequentie, shift en CTCSS ingesteld. Deze kunnen worden overschreven met de Shift en Tone functies zoals beschreven bij Frequentie mode.

De scanknop scant de lijst van repeaters. In de lijst van repeaters wordt regelmatig de frequentie meerdere keren genoemd. Dit omdat de betreffende frequentie voor meerdere repeaters wordt gebruikt en de bijbehorende CTCSS-code kan afwijken. Een goed voorbeeld zijn Nijmegen, Groningen en Zoetermeer. Alle drie op kanaal 140 (weet je het nog: (140*12,5) + 144000 = 145750).

De mode MEM (Memory)

Hierbij loop je met de pijltjes door de lijst van memory kanalen. Er zijn 10 kanalen (0 t/m 9) beschikbaar. Dit kan natuurlijk best meer zijn, maar ik vond het voor mijzelf ruim voldoende. Mocht je er iets van vinden, in de software is het eenvoudig aan te passen. De memory kanalen dienen wel gevuld te worden. Dit doe je door de gewenste frequentie, shift, CTCSS te kiezen of de gewenste repeater. Druk vervolgens op Save op pagina 3. Druk op het gewenste memory-nummer (0 t/m 9) en druk op enter. De scanknop scant de lijst van geheugenkanalen.

Standaard staat één van de 3 modi actief en is deze te zien in de pijltjesknop en is betreffende knop highlighted. Als op een van de knoppen Vol, SQL, Scan, Tone, Light wordt gedrukt, wordt deze knop gedurende 10 seconden actief. Daarna schakelt de transceiver weer terug naar Freq, RPT of MEM.

De knoppen en functies

De knoppen zijn niet alleen knoppen maar voorzien je ook van de nodige informatie: De knop is horizontaal in 2 gedeeld, bovenin de functie en onderin de gekozen waarde als dit van toepassing is. De bovenkant is highlighted als de functie actief is. Een functie kan actief zijn als deze met de pijltjes kan worden bediend. De kleur onderin kan ook afwijken om je duidelijk te maken dat een functie is geactiveerd. Voorbeelden daarvan zijn de Scan, deze is rood als er wordt gescanned en de Shift, deze is rood als de shift -600kHz is en groen als de shift +600kHz is.

Pagina1:

Vol: Volume, 0 t/m 8, 0 is mute.
SQL: Squelch, 0 t/m 8, 0 is helemaal open.
Scan: Activeer of stop het scannen in de geselecteerde mode (Freq, RPT of MEM) Als de scan knop actief is, kan met de pijltjes de scanmode (auto resume of stop) worden gekozen.
Off: Zet de transceiver niet echt uit maar standby, het display en de radio gaan uit. Door op een willekeurige plek op het scherm te drukken, start de transceiver opnieuw op.
Freq: Selecteert de Frequentiemode.
RPT: Selecteert de repeater mode.
MEM: Selecteert de memory mode.
Next: Ga naar pagina 2.
Pijltes: Vorige en volgende, afhankelijk van de actieve functie.

Pagina 2:

Shift: Selecteer + of – 600KHz shift.

```
typedef struct {
  uint16_t rxChannel;    // RX Freq channel (12.5KHz steps, 144.000MHz = 0, channel 128 is 145.600MHz)
  uint16_t txChannel;    // TX Freq channel (12.5KHz steps, 144.000MHz = 0, channel 128 is 145.600MHz)
  uint8_t repeater;      // Repeater number from the list of repeaters, overwrites RX and TX channel
  int8_t txShift;        // Shift + or -
  byte hasTone;          // 0=disabled, 1=only RX, 2=only TX, 3=RX and TX
  byte ctcssTone;        // See list of CTCSS codes in config.h
} Memory;
```

Mute: Mute de audio. Druk nogmaals om de mute op te heffen of verander het volume met de volume functie.
Tone: Selecteer de gewenste CTCSS-functie: R(X), T(X) en RT (RX en TX). Met de pijltjes kan de gewenste tone worden geselecteerd.
Reverse: Schakel de RX en TX-frequentie om als de shift actief is. (Luister op de ingang van de repeater).
Prev: Vorige pagina.
MOX: Activeer de zender.
APRS: Zet de APRS-functies aan of uit (zie verder).
Next: Volgende pagina.
Beacon: Verstuur een APRS-baken. Als de transceiver is gekoppeld met het internet wordt het baken verstuurd via internet, anders via de radio.
TX Beacon: Verstuur een APRS-baken via de radio.
Pijltes: Vorige en volgende, afhankelijk van de actieve functie.

Pagina 3:
Power: Schakelt tussen laag en hoog vermogen, 0,5 of 1 watt uit de standaard DRA818.
Band: Schakel tussen UHF en VHF. Dit is niet zinvol, de DRA818 is op 430MHz te doof.
Light: Regel het niveau van het backlight.
Calibrate: Zie bij software onder Display.
Prev: Vorige pagina.
Save: Bewaar ingestelde frequentie, shift en CTCSS-tone of geselecteerde repeater in het geheugen. Na het indrukken kies je het gewenste geheugenkanaal.
Print: Print de instellingen naar de seriële USB-poort van de ESP32.
Next: Volgende pagina.
Pijltes: Vorige en volgende, afhankelijk van de actieve functie.

Pagina 4:
Op pagina 4 zijn de dynamische gegevens van de GPS-ontvanger en de Wi-Fi te vinden. Tenminste, als de transceiver is gekoppeld met Wi-Fi. Dit is vooral handig als je het IP-adres van de transceiver wilt weten. Door op de knop Close te drukken ga je terug naar pagina 1.

De meter

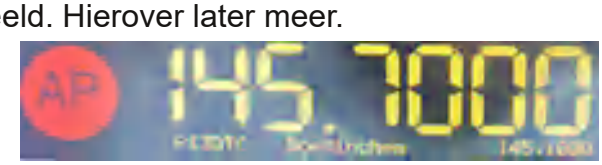
Hier moet ik een bekentenis doen: de meter is niet geïmplementeerd. Wel in de software maar hij zit nergens aan vast, simpel omdat de DRA818 daar geen faciliteiten voor heeft.

De meter wijst simpel S9+10dB aan als de squelch open is en bij het zenden staat die op P4 bij laag en P9 bij hoog vermogen. Frank PA3CNO had nog wel een leuke suggestie: haal de audio door een hoog doorlaat filter, want hoe meer ruis hoe meer hoog er in het signaal zit. Richt dit gelijk en meet dit. Leuk voorstel, maar ik ben er nog niet aan toe gekomen.

De frequentieweergave

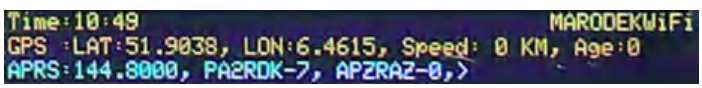
Hier wordt uiteraard de gebruikte RX-frequentie getoond en klein eronder de TX-frequentie.

Als de RPT-mode is geactiveerd, staan er ook de call en plaats van de repeater. De kleur van de tekst is geel. Als de squelch open is, wordt de kleur groen en als er wordt gezonden is de tekst rood en komt er een bolletje met TX in beeld. Als er via Wi-Fi een APRS-baken wordt verzonden, komt er een rood balletje met de tekst AP in beeld. Hierover later meer.



De statusregels.

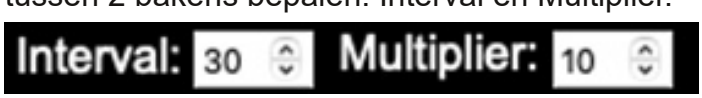
Boven in het scherm staan verschillende regels; de bovenste regel toont de tijd, maar alleen als er een GPS-signaal is, en rechts de SSID van de Wi-Fi, tenminste als de transceiver met Wi-Fi is verbonden. De tweede regel toont de GPS-informatie Lat, Lon, Speed en de ouderdom in mSec van de informatie. Deze regel is geel als de GPS valide informatie heeft en rood als de informatie niet valide is, bijvoorbeeld als de actuele positie nog niet is gevonden of ouder is dan 5 seconden.



De derde regel toont de APRS-instellingen Frequentie, Call, Destination en symbool zoals ingesteld in de transceiver. Maar alleen als de APRS-functie aan staat. Zie bij knoppen, pagina 2.

APRS

Het heet niet voor niets een APRS transceiver: wie de eerdere verhalen heeft gelezen weet ook hoe deze tot stand is gekomen. De transceiver kan automatisch APRS-bakens verzenden. Daartoe schakelt hij regelmatig naar de ingestelde APRS-frequentie en verstuurt een baken, gebruik makend van de ingestelde parameters. Zie ook instellingen.
In de instellingen zijn er 2 parameters die de tijd tussen 2 bakens bepalen: Interval en Multiplier.



Bij stilstand is de tijd Interval*Multiplier, hier dus 300 seconden, 5 minuten. Als de snelheid boven de 5km/h komt is de tijd interval*4. Boven de 25km/h is deze interval*2 en boven de 80km/h is de tijd interval*1 (hier dus 30 seconden).

De transceiver is voorzien van smart-beaconing, dit wil zeggen dat als er een koerswijziging van meer dan 27° is, dus bij vrijwel elke bocht en op een rotonde, meerdere malen een baken wordt verstuurd.

Het baken wordt alleen verstuurd als er niet wordt gezonden, de squelch dicht is, de GPS-informatie actueel is en de APRS-functie aan staat.

In de instellingen kan worden ingesteld dat er altijd een baken wordt verstuurd direct nadat de PTT wordt losgelaten. Dit zorgt ervoor dat ook in een lopend QSO er zo nu en dan een baken wordt verstuurd.

Maar…

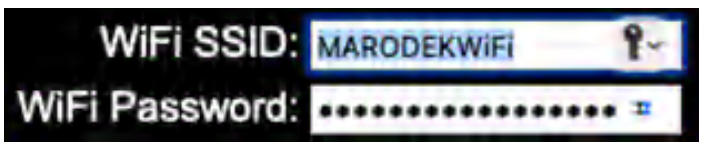
In de transceiver zit een ESP32 met de mogelijkheid om te verbinden met een Wi-Fi netwerk. Hiervan maak ik uiteraard gebruik. Als de transceiver met Wi-Fi is verbonden en toegang heeft tot het internet, worden de bakens via Wi-Fi verstuurd. Niet echt ‘radioamateur’ achtig natuurlijk maar wel heel functioneel. Hiervoor is wel een APRS-gateway noodzakelijk. Deze is daarom ook in de software gebouwd.

Zie je in het plaatje van aprs.fi op bladzijde 2 dat mooie lijntje van Sinderen naar Oosterhout met al die rode stippen (bakens)? Ik heb in de auto de transceiver met de hotspot van mijn telefoon verbonden. Dan heeft de transceiver dus internet beschikbaar en worden alle bakens via internet verstuurd. Niet afhankelijk van de bereikbaarheid van APRS-gateways onderweg, dus (bijna) 100% dekking.

Wi-Fi

De transceiver kan verbinden met Wi-Fi en is dan via een browser bereikbaar. Maar initieel moeten jouw Wi-Fi gegevens dan worden ingesteld. Dit gaat als volgt: Als de transceiver niet kan verbinden met een Wi-Fi netwerk, wordt het zelf een access-point. Met PC, tablet of telefoon kan je verbinding maken met dit access-point.

Deze heet APRSTRX. Als de verbinding tot stand is gekomen, ga dan in de browser naar IP-adres 192.168.4.1. Je hebt nu toegang tot de transceiver en kan hem volledig via de browser bedienen. Onderin zie je een knop ‘settings’, hier kunnen alle instellingen van de transceiver worden aangepast, maar hierover later meer. Nu gaat het alleen om de eerste 2 velden.



Geef hier het SSID en Password van jouw Wi-Fi op. Druk onderin op Submit Druk nu onderin op Reboot

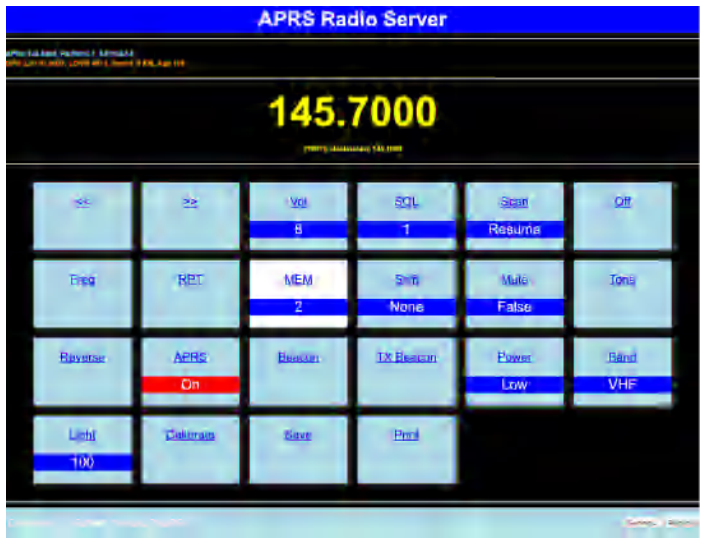
De transceiver start opnieuw op en maakt, als alles goed gaat, verbinding met jouw Wi-Fi. Dit kan je zien als in het display op de transceiver rechtsboven de naam van jouw Wi-Fi staat.

Verbind de PC, tablet of telefoon weer met jouw Wi-Fi.

Ga naar pagina 4 op het display, hier zie je wederom de SSID en belangrijker, het IP-adres van de transceiver.

Ga nu in jouw browser naar het genoemde IP-adres, in dit voorbeeld 192.168.5.140 en je hebt toegang tot de transceiver en ook de settings pagina.

Zoals al genoemd, de transceiver is volledig te bedienen via de browser, alle knoppen en



De ingebouwde webserver

informatie zoals op het display zijn ook beschikbaar in de browser. Alleen de MOX heb ik uitgezet, om te voorkomen dat de transceiver per ongeluk op zenden wordt gezet. Op het verlanglijstje heb ik de mogelijkheid nog staan om audio te streamen van en naar de transceiver, zodat deze echt helemaal remote te bedienen en gebruiken is, maar zover is het nog niet.

De transceiver is voorzien van de multi-wifi library. Dit maakt het mogelijk om te verbinden met verschillende Wi-Fi netwerken. Om deze functionaliteit te gebruiken, dien je in de config.h de verschillende netwerken te configureren. Deze mogelijkheid is dus niet beschikbaar in de settings. In de settings kan slechts één netwerk worden geconfigureerd. Deze wordt toegevoegd

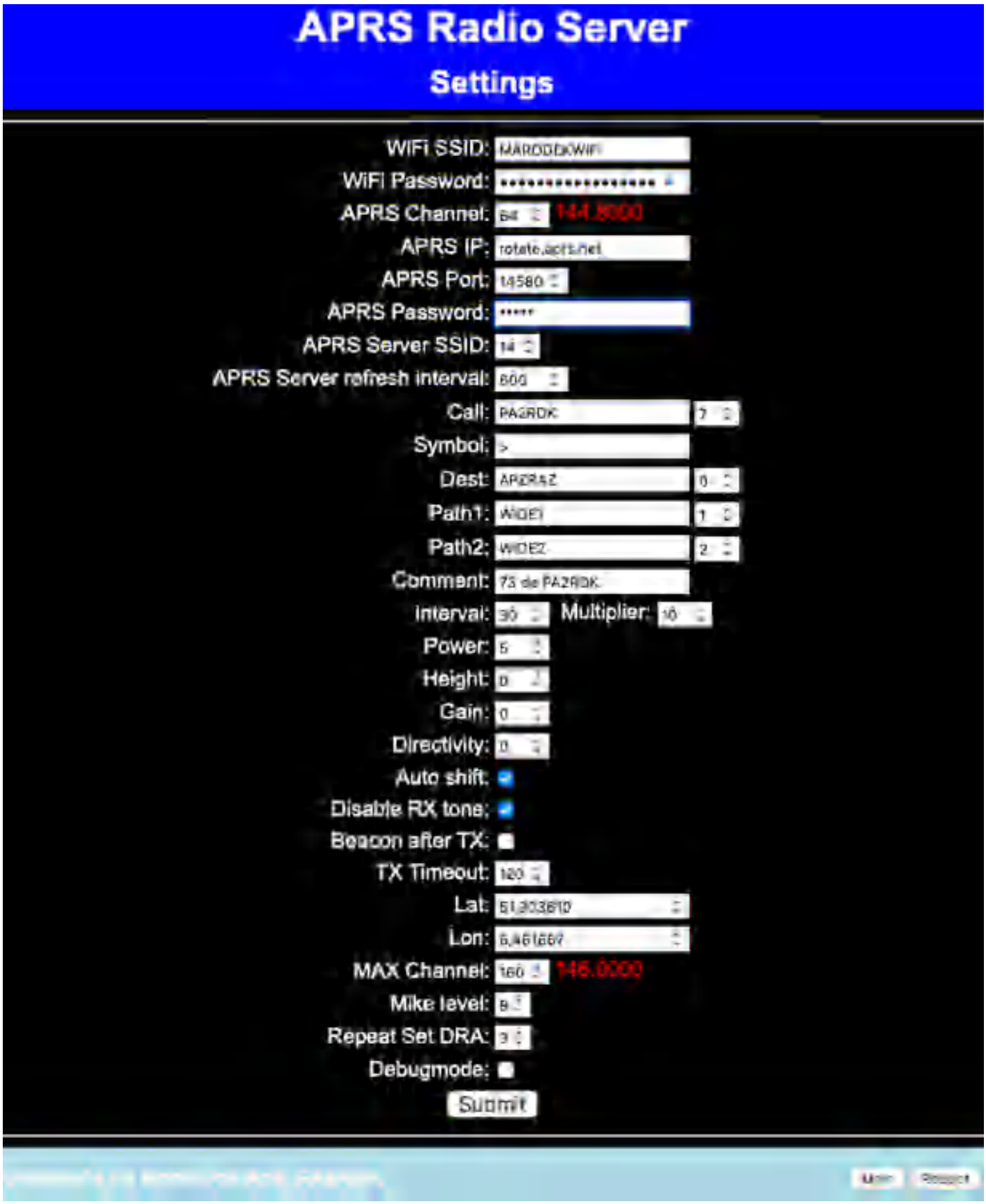
```
const wlanSSID wifiNetworks[] {
    {"PI4RAZ","**********"},
    {"Loretz_Gast", "********"}
};
```

Settings

De instellingen, in goed Engels settings, kunnen alleen worden ingesteld via een Wi-Fi verbinding, zoals beschreven bij Wi-Fi. Mocht je het fenomeen internet helemaal hebben gemist, of lukt het niet de Wi-Fi in te stellen zoals hierboven beschreven, dan kun je de instellingen

ook aanpassen in de config.h. Alle instellingen zitten in één struct ‘Settings’. Deze wordt gedefinieerd in aprstrx.ino en de initiële waardes staan in config.h. Maar nogmaals, de juiste weg is via de browser.

Wat betekenen de instellingen? Lees ook het verhaal in de Razzies van juni 2019 over de iGate eens door, daar wordt dieper ingegaan op de materie.



WiFi SSID en WiFi Password: heb ik hierboven besproken, dit zijn de wifi-instellingen voor jouw Wi-Fi.

APRS Channel: De gebruikte APRS-frequentie. Standaard in Nederland is dat 144.800

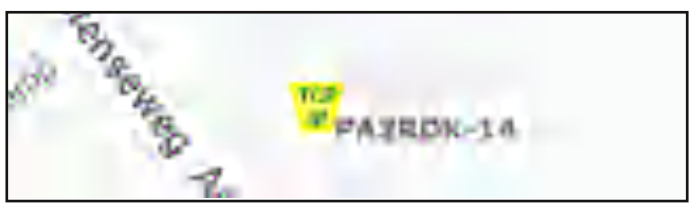
APRS IP, APRS Port, APRS Password en APRS Server SSID: De instellingen voor de ingebouwde APRS-gateway.

APRS IP: De URL waar de iGate zich moet aanmelden. Default rotate.aprs.net.

APRSPort: Het poortnummer dat hij moet gebruiken voor het aanmelden. Default 14580

APRS Password: Jouw gegenereerde APRS Key (maximaal 6 karakters). Deze kun je genereren op https://aprs.do3sww.de.

APRS Server SSID: De gateway wordt ook op aprs.fi getoond en met deze SSID en de call, dus in dit voorbeeld met PA2RDK-14



APRS Server refresh interval: De gateway dient zich regelmatig te melden op aprs.fi. In dit voorbeeld dus elke 600 seconden, 10 minuten.

De volgende settings gaan over APRS zelf:

Call: De gebruikte call en SSID, hier dus PA2RDK-7.

Symbol: Het op APRS.FI getoonde symbool (> = auto).
Zie ook http://www.aprs.org/symbols/symbolsX.txt.

Dest: De bestemming van het packet. Dit moet altijd beginnen met AP om als APRS packet herkend te worden. In dit geval APZRAZ-0.

Path 1: Bepaalt het pad voor relay stations. Vroeger stond hier RELAY, maar sinds een aantal jaren wordt hier WIDE1-1 gebruikt

Path 2: Bepaalt het pad voor de digipeaters. Dit is altijd WIDE, en het eerste getal geeft aan langs hoeveel digipeaters het packet mag reizen voor hij weggegooid wordt. Maak hier nooit meer dan 3 van, omdat je dan het netwerk onnodig belast. 2 of 3 is prima om weergegeven te worden op de kaart. Hier dus WIDE2. Het tweede getal wordt elke keer verminderd als een repeater gepasseerd wordt. Initieel gelijk aan het aantal digipeaters in de WIDE definitie, dus hier weer 2. Het volledige path wordt daarmee WIDE2-2

Comment: Stukje vrije tekst dat in de tekstbox op de kaart verschijnt.

Interval & multiplier: De frequentie waarmee de bakens worden verstuurd, zie onder APRS.

Power: Het vermogen van de transceiver.

Height: De hoogte van de antenne.

Gain: De versterking van de antenne.

Directivity: De richting van de antenne.

De overige instellingen zijn diversen:

Auto shift: Tussen 145.575MHz en 145.800MHz staat de shift (-600kHz) automatisch aan.

Disable RX tone: Ondanks dat de CTCSS op R of RT staat, wordt de RX CTCSS niet gebruikt. Dit is vooral in repeater mode handig, omdat daar de CTCSS hard wordt ingesteld.

Beacon after TX: Als de PTT wordt losgelaten, wordt er een APRS-baken verzonden, als de APRS aan staat en er een GPS-fix is.

TX Timeout: Na (in dit geval) 120 seconden stopt de transceiver automatisch met zenden. Als hier 0 staat, werkt de time-out niet.

Lat en Lon: Als er geen GPS is aangesloten of er geen GPS-fix is en debug in settings staat aan, dan worden er APRS-bakens verstuurd met deze positie.

MAX Channel: Als je 'buiten de band' wilt ontvangen (of zenden) verhoog dan dit getal met de formule (gewenste frequentie – 140000)/12,5. Dus bijvoorbeeld 148000-144000=4000, 4000/12,5=320.

Mike level: Het microfoon niveau van de DRA, op een schaal van 0 t/m 8.

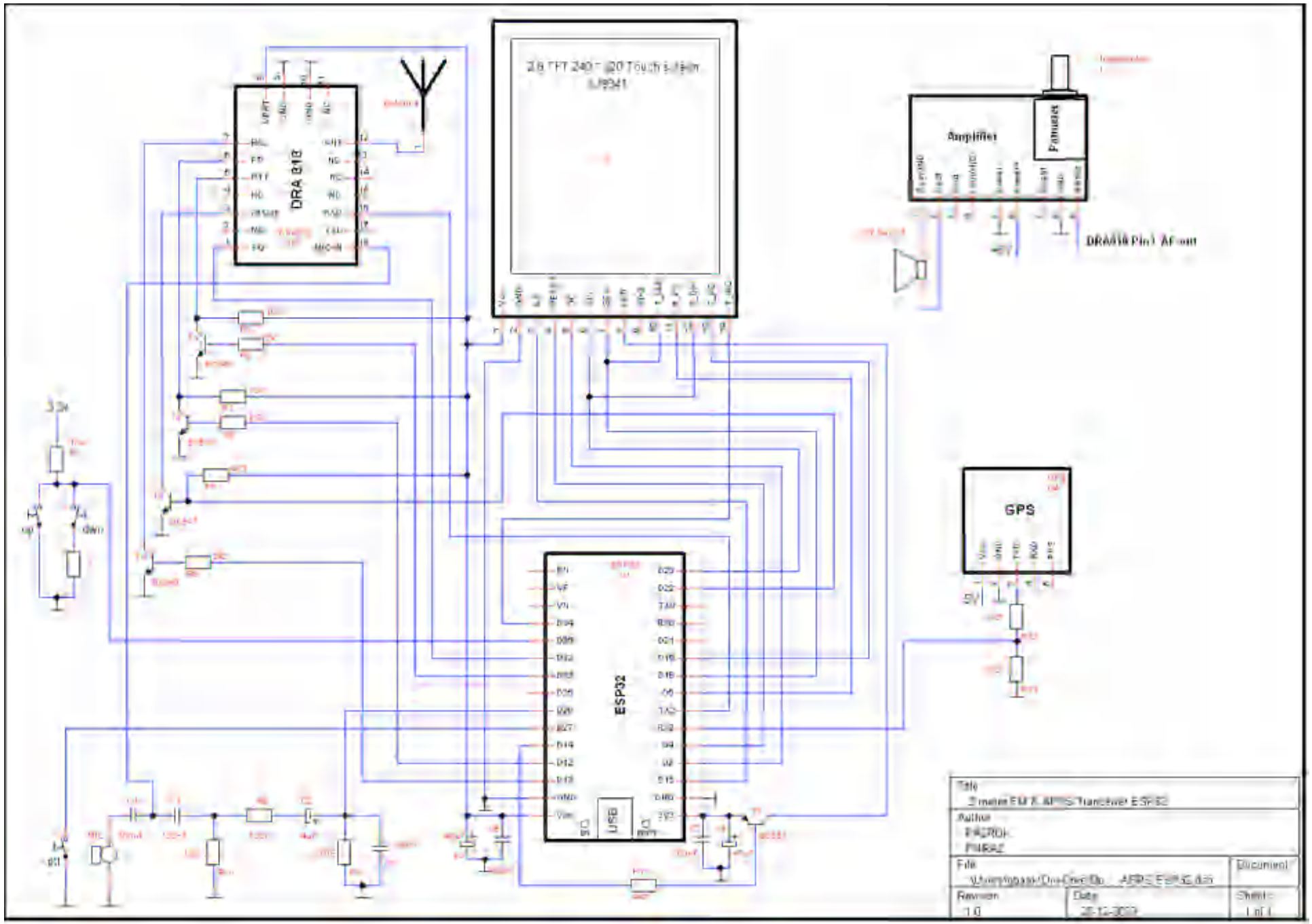
Repeat Set DRA: Sommige DRA's zijn wel eens ongehoorzaam en luisteren niet zo goed. Daarom kunnen de DRA commando's automatisch worden herhaald. In dit geval wordt elk commando 3 keer verstuurd.

Debugmode: Zonder GPS of GPS-fix worden er toch APRS-bakens verstuurd. Ook wordt er via de USB RS232 van de ESP32 debug informatie naar een terminal gestuurd.

Het schema

Het schema is niet heel ingewikkeld, de basis bestaat uit een DRA818, dit is een complete VHF transceiver met een RS232 interface. Een SA818 zou het ook moeten doen, maar die heb ik nog niet en heb ik dus nog niet kunnen testen.

Alle logica zit in een ESP32 die met de Arduino software kan worden geprogrammeerd. Het display is een 240x320 kleurenscherm met touch functionaliteit. Zoek op Ali op de ILI9341. Let wel op de grootte en of er touch op zit!


DRA 818
2.8 TFT 240*320 Touch Shield
ILI9341
Amplifier
Potmeter
GPS
ESP32
USB
up
down

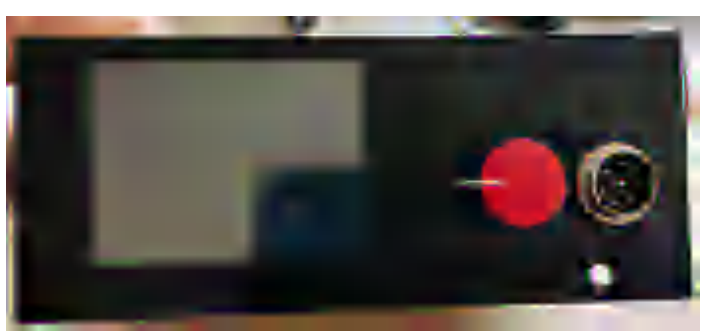
Ook zit er een GPS-ontvanger op, noodzakelijk voor de APRS-positie.
Tenslotte een audio versterker. Ik heb er een genomen met volumeregeling en een on/off schakelaar.
T1 schakelt vanuit pin D33 van de ESP de PTT.
T2 activeert vanuit pin D12 de sleep mode van de DRA.
T3 schakelt vanuit pin D22 de mute van de versterker door de ingang naar massa te trekken. Deze zorgt ervoor dat de radio stil is als de squelch dicht is en dat de mute functie werkt. De DRA kan het volume niet naar 0 regelen, daarom zorgt T3 er ook voor dat als het volume op 0 staat, de mute aan staat.
T4 schakelt vanuit pin D13 tussen hoog- en laagvermogen.
T5 regelt vanuit pin D14 de intensiteit van het backlight van het display.
De microfoon zit via een C van 100nF direct aan de DRA en wordt via een filter en verzwakker gemengd met het APRS-signaal uit de ESP pin D26.
RX2 en TX2 van de ESP worden gebruikt door de GPS en DRA. De TX stuurt commando's naar de DRA, de RX ontvangt data uit de GPS-ontvanger. Gelukkig draaien ze beiden op dezelfde baudrate. Op deze manier is er maar één RS232 poort noodzakelijk voor beide devices.

Dit kan omdat de GPS geen belangstelling heeft voor informatie uit de ESP en andersom de DRA niet terugpraat als hij een commando ontvangt.
Als ik nog eens met een SA818 ga spelen moet dit anders, want de SA praat wel terug met nuttige informatie.
Pin D36 wordt gebruikt in analoge mode om de up en down knoppen van de microfoon uit te lezen. Standaard is de pin hoog, als Up wordt ingedrukt wordt de pin laag en als de Down wordt ingedrukt komt D36 ongeveer op de halve spanning. De weerstand R in het schema is dus, net als R7, 10k.
De GPS draait op 5 Volt, daarom wordt de TX vanuit de GPS via een spanningsdeler aangeboden aan de ESP, want deze draait op 3.3V.

De bouw

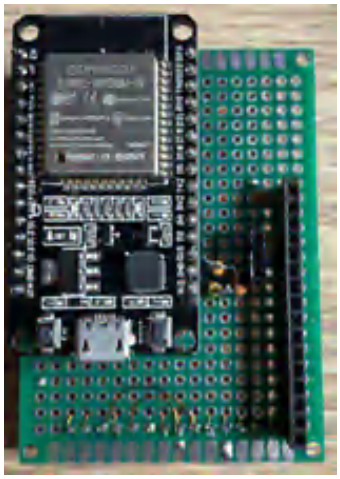
De aanleiding om te starten met dit project was de aankoop van een oude defecte Philips counter op de verkoping van de Veron afdeling in Doetinchem. Voor de counter had ik geen belangstelling, maar het kastje vond ik heel fraai.

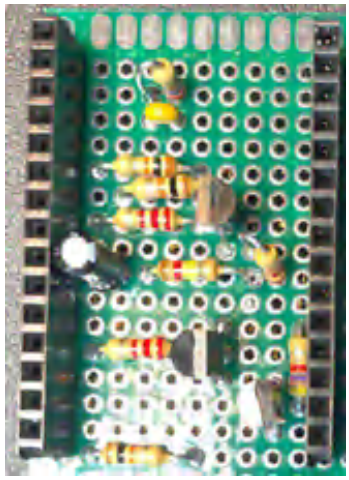
Het kastje bestaat uit een metalen frame dat wordt geschoven in een plastic kast. Het front is natuurlijk op geen manier bruikbaar, dus heb ik een nieuw front gemaakt met de 3D printer en het frame enigszins aangepast.



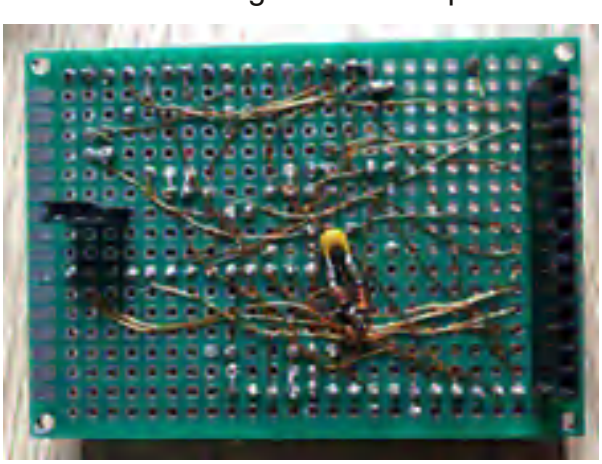
Als bodemplaat heb ik een stuk dubbelzijdig printplaat gebruikt en zoals goed te zien op de foto, heb ik nog een constructie geprint om de schakeling in te monteren.

Het display zit uiteraard tegen het front gemonteerd en als piggyback print zit hier de print tegen waarop de ESP32 huist. De ruimte onder de ESP32 is natuurlijk ook niet onbenut, hier zitten een aantal van de schakeltorren.

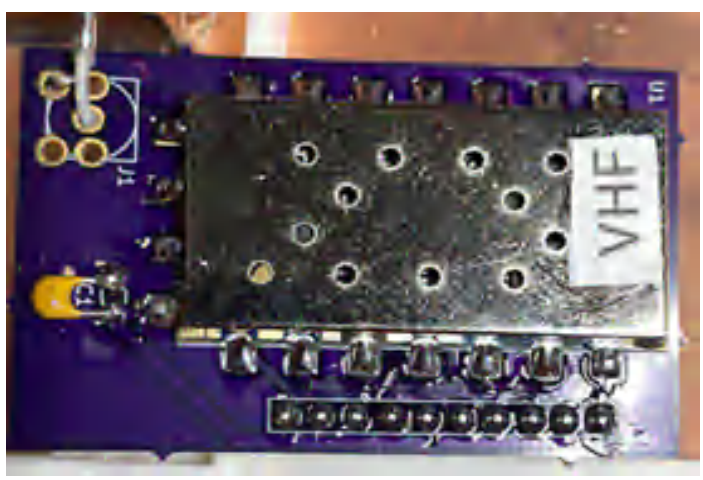




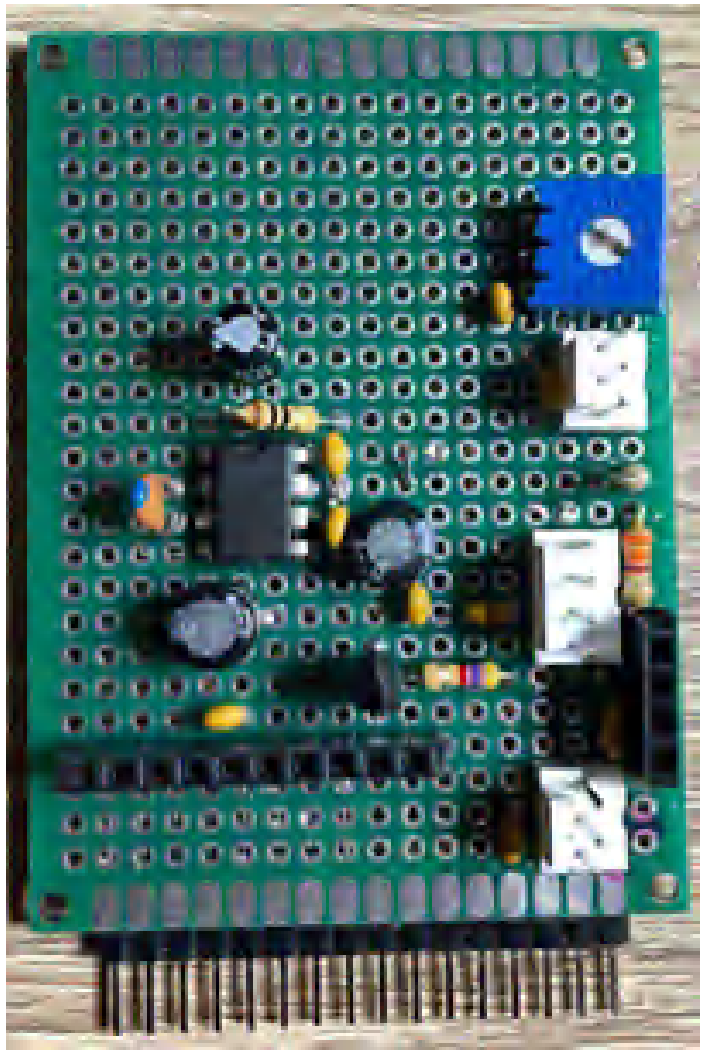
Het is in de RAZzies al vaker ter sprake geweest, ik vind het fijn om de printen te bedraden met dun geïsoleerd koperdraad. Het neemt weinig ruimte in en laat zich goed vertinnen. Goed licht en een loep zijn wel onontbeerlijk (tenminste op mijn leeftijd).



Met de connector die aan de koperzijde van de print zit, wordt deze tegen het display gedrukt. Aan de componentzijde zit ook een connector, deze is gebruikt voor een tweede print. Hierop zit de DRA818 bevestigd. Een DRA818 heeft een onhandige layout om op een stukje gaatjesprint te gebruiken, maar gelukkig heeft Oshpark fraaie breakout boards, zodat ze in een printheader geprikt kunnen worden, of rechtstreeks op een gaatjesprint gesoldeerd. Voor een paar dollar heb je er 3, kan je er ook nog eens een weggeven.



Op deze print heb ik ook de connectoren voor de voeding, de audioversterker, de microfoon-plug en de GPS gezet. De overige onderdelen zijn een LM386 audioversterker en bijbehorende onderdelen. Deze staan niet op het schema en worden ook nog niet gebruikt.

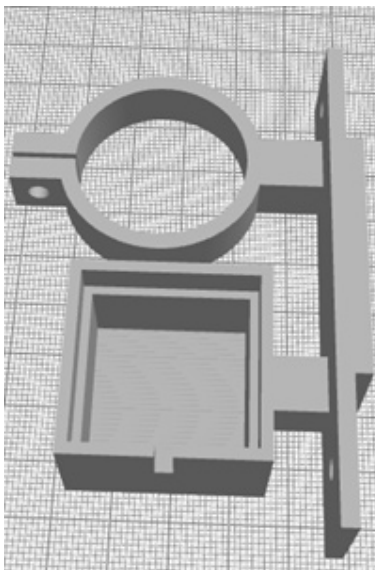


Het is de bedoeling in de toekomst deze versterker te gebruiken om de audio uit de DRA terug te koppelen naar de ESP32 en op deze wijze de audio te streamen. Ook moet het worden gebruikt om de transceiver als APRS-gateway te kunnen gebruiken. Alles in elkaar en op zijn plaats ziet er als volgt uit.

Voor de speaker en de GPS-antenne heb ik ook een steuntje ontworpen en geprint. Deze is gemonteerd aan het frame en hangt boven het voedingsprintje.

Dat brengt mij bij de voeding, in het kastje zat al een trafo en een bescheiden voeding ingebouwd. Van de componenten van de oorspronkelijke voeding heb ik een ouderwetse analoge 5V voeding met een 7805 gemaakt. Wederom met een 3D geprint frame in het kastje gebouwd en de 7805 tegen het frame aan, dan moet het met de koeling wel goed komen. Toch

leuk een setje dat op 230V kan werken. Maar de transceiver is bedoeld voor in de auto, dus heb ik ook een 12V connector op de achterkant gezet.



Voor de audio maak ik gebruik van een standaard 3W versterker te koop bij Ali. Deze is al voorzien van een potmeter met schakelaar, lekker makkelijk leek mij. Dat viel een beetje tegen: de schakelaar in de potmeter schakelt de versterker. Toen ik probeerde de geschakelde voedingsspanning uit de versterker voor de rest van de schakeling te gebruiken, ging het helemaal mis. Eerst ging het oscilleren en vervolgens blies ik de versterker op. Natuurlijk wilde ik zeker weten dat het aan mij lag, dus heb ik een andere versterker gepakt (ik had er gelukkig een aantal gekocht) en probeerde ik het nogmaals. Dat was versterker nummer 2. Maar een defecte versterker was wel ergens goed voor: die kon ik mooi gebruiken om uit te zoeken of ik de schakelaar niet los kon koppelen van de versterker. Dat bleek mogelijk door de C van 470uF los te halen, daaronder loopt het printspoor van de schakelaar. Deze doorsnijden, de voeding omleggen, de C weer terugplaatsen en de schakelaar is geëlimineerd uit de schakeling. Best een gepiel, ik denk dat het makkelijker is om een versterker zonder potmeter te kopen en een losse potmeter met schakelaar te gebruiken. Of natuurlijk alleen een losse schakelaar gebruiken.

Een DRA818 levert standaard 0.5 of 1 Watt. Ik vind dat een beetje krap. In de webshop van PI4RAZ staat echter een heel mooi klein eindtrapje, voorzien van een VOX, te koop die van de 0.5 Watt graag 7 Watt maakt. Na een sponsoring aan de penningmeester van RAZ ;-) ontving ik de eindtrap. Deze heb ik ook in het kastje gebouwd en netjes ingeblikt met behulp

van stukjes printplaat. De versterker heb ik tegen het frame aan geschroefd, dat bespaart de (ruimte voor) een koellichaam en in de praktijk is het voldoende koeling. Toen het allemaal dicht zat, realiseerde ik mij dat ik vergeten was foto's te maken. Sorry, maar ik maak hem niet meer open.

De eindtrap werkt op 12V en vraagt een beetje teveel van de ingebouwde voeding, daarom werkt de eindtrap alleen op 12V en niet als er gebruik wordt gemaakt van de ingebouwde voeding. Een paar diodes helpen daarbij.

Het geheel ziet er dan uit zoals te zien op de foto op de volgende bladzijde. Hierbij is links boven de afscherming van de eindtrap te zien. Het complete frame schuift zo in het kastje en met 2 schroeven achterin zit die vast. Restte er nog een ding, met de speaker in de kast hoor je niet veel, dus heb ik wederom met 3D printer een rooster geprint en met de gatenzaag een

gat in de kast geboord. Ziet er best goed uit vind ik.

De bouwbeschrijving is natuurlijk niet bedoeld om exact na te bouwen. Ik denk dat het al een flinke uitdaging wordt om aan het door mij gebruikte kastje te komen. Maar het geeft wellicht wel leuke ideeën als je met deze transceiver aan de gang wilt gaan. Mocht je toevallig wel aan dit kastje kunnen komen, dan staan alle 3D modellen in de GitHub pagina van dit project. Deze vind je hier.

Vooralsnog is er geen print van het project. Mocht daar belangstelling voor zijn, dan horen we het graag. We beloven natuurlijk niets, maar je weet nooit.

De transceiver in de finale vorm. Het witte puntje onder de microfoonconnector is een stift die uit het front getrokken kan worden en waarmee het display bediend kan worden.

Met bovenstaande informatie en als je in staat bent een ESP32 te programmeren met de Arduino omgeving, zou het mogelijk moeten zijn de transceiver te bouwen. Dus hier zou dit artikel kunnen stoppen. Maar er is natuurlijk nog van alles te schrijven over de…

Software

Maak je geen zorgen, dit wordt geen cursus programmeren en ik ga ook niet de code regel voor regel uitspellen. Wel ga ik ervan uit dat je mijn verhaal in de RAZzies van februari hebt gelezen en dat je beschikt over de nodige kennis van het programmeren in de Arduino omgeving. Natuurlijk kan ik het niet nalaten hier en daar een belerend vingertje op te steken, je moet me maar vergeven.

Het besturen van de DRA is niet heel ingewikkeld en ook het uitlezen van een GPS is niet zo ingewikkeld. Wel is het ontwikkelen van een goede userinterface lastig (tenminste voor mij). Daarbij voorzag ik dat, eenmaal aan het ontwikkelen, de wensen en eisen nog wel eens

kunnen wijzigen. De interface moet dus flexibel aan te passen zijn. Daarbij is het ook een uitdaging om op het scherm alle statusinformatie te plaatsen, zonder dat het een rommeltje wordt. Denk hierbij aan de shift, de CTCSS, volume en squelch level etc. De transceiver moet min of meer intuïtief te bedienen zijn.

De knoppen

Mede geïnspireerd door de layout van mijn FT-991A bedacht ik dat een knop waarin (zo nodig) ook de status wordt vermeld een mooi oplossing kan zijn. Ik ben begonnen met een struct met daarin alle parameters van een knop.

```
typedef struct {  // Buttons
  const char *name;      // Buttonname
  const char *caption;   // Buttoncaption
  char waarde[12];       // Buttontext
  uint16_t pageNo;
  uint16_t xPos;
  uint16_t yPos;
  uint16_t width;
  uint16_t height;
  uint16_t bottomColor;
  uint16_t topColor;
} Button;
```

Name en Caption zijn constante, deze zijn dus runtime niet te wijzigen. In ‘waarde’ staat de tekst die onderin de knop komt te staan (de status). ‘pageNo’ is de pagina waarop de knop moet worden getoond.

xPos, yPos, width en height bepalen grootte en positie van de knop. De topColor is de kleur van het bovenste deel van de knop, de bottomColor is de kleur van het onderste deel, waarop de ‘waarde’ staat. Als je dus een ander scherm wilt gebruiken is dit de aangewezen plek om de knoppen aan te passen.

De verschillende knoppen heb ik in een array ‘buttons[]’ geplaatst. De functie ‘DrawButtons’ loopt door de array van knoppen en roept per knop de functie void DrawButton(String btnName) aan. Deze functie overrides de functie void DrawButton(int xPos, int yPos, int width, int height, String caption, String waarde, uint16_t bottomColor, uint16_t topColor, String Name). Inderdaad, hierbij worden alle variabelen van de knop als parameter meegegeven. De reden hiervoor is dat er ook knoppen kunnen worden getekend die niet in de array staan. Dit gebeurt bijvoorbeeld bij het opstarten als de WiFi wordt gezocht. Er gebeurt nog een leuk grapje in DrawButton(String btnName), Hierin wordt met de functie Button FindButtonByName(String Name) de volledige knop met al zijn variabelen opgezocht en vervolgens wordt hierin de functie Button FindButtonInfo(Button button) aangeroepen. Dit is de functie waarin de gegevens van de knop kunnen worden aangepast. Dit gebeurt dus consequent als de button wordt getekend. Je kunt dus, als je het leuk vindt of er een nuttige toepassing voor weet te verzinnen, de knoppen runtime verplaatsen.

In het programma bestaat een globale variabele ‘actualPage’. Deze wordt gebruikt om te bepalen welke pagina wordt getoond. Maar omdat sommige knoppen op meerdere pagina’s voorkomen, wordt de functie int ShowControls() gebruikt om de juiste pagina en bijbehorende knoppen te tonen. Je kunt ook knoppen op elkaar leggen, in de afhandeling wordt dan de code van beide knoppen uitgevoerd. Hiervan maak ik bij de pijltjes gebruik.

De afhandeling

De software heeft het niet erg druk, daarom heb ik geen interrupts hoeven toepassen bij het afhandelen van de knoppen. In de loop kijk ik met bool pressed = tft.getTouch(&x, &y); regelmatig of het display is aangeraakt. Zo ja dan komen in x en y de positie waar het display is ingedrukt. In de array van knoppen kijk ik welke knop x en y omhult. Vervolgens roep ik HandleFunction(buttons[i], x, y) aan, waarbij buttons[i] de betreffende button is en x en y de positie waar er is gedrukt. Deze x en y heb ik alleen nodig bij de pijltesknop om te bepalen

waar er op de knop is gedrukt. Dit om de stapgrootte van de frequentie te kunnen bepalen. De functie void HandleFunction() kent 5 overrides. De reden hiervoor is dat ik de functie niet alleen aanroep als het display wordt aangeraakt, maar ook vanuit de webpagina en bij gebruik van de knoppen op de microfoon. In HandleFunction wordt alle gebruikersinteractie afgehandeld. Afhankelijk van de gekozen functie worden er een of meer of zelfs alle knoppen opnieuw getekend. Want hoe minder er wordt hertekend, hoe rustiger het display. Als je iedere keer het hele display opnieuw tekent, wordt het een nerveus knipperend geheel. Een leuk voorbeeld is de Reverse knop. Alleen als RX en TX ongelijk zijn, worden deze omgedraaid, wordt de frequentie opnieuw getekend en worden de Shift en Reverse knop opnieuw getekend.

Ook het toetsenbord is een pagina die op dezelfde wijze wordt afgehandeld als alle andere knoppen. Het voordeel van deze werkwijze is dat een functie altijd op maar een plek wordt uitgevoerd. Is die fout dan is die altijd fout, maar als die goed is dan werkt het ook altijd. En, je herhaalt geen code.

```
if (button.name == "Reverse") {
  if (settings.rxChannel != settings.txChannel) {
    isReverse = !isReverse;
    SetFreq(0, 0, 0, false);
    if (doDraw) DrawFrequency(false);
    if (doDraw) DrawButton("Shift");
    if (doDraw) DrawButton("Reverse");
    delay(200);
  } else isReverse = false;
}
```

De instellingen

In het programma maak ik gebruik van een struct ‘Settings’. Hierin zitten alle instellingen maar ook alle variabelen die bewaard moeten worden als de transceiver uit wordt gezet, zoals bijvoorbeeld de laatst gebruikte mode en frequentie. De lol van zo’n struct is, dat het eigenlijk een array van bytes is. Deze array kan eenvoudig in de EEPROM worden gezet en worden opgehaald. Elke 10 seconden check ik de inhoud van Struct per byte. Als deze gewijzigd is dan schrijf ik hem in de EEPROM. Dat checken wordt gedaan om te voorkomen dat we elke 10 seconden hetzelfde wegschrijven en zodoende de levensduur van de EEPROM flink zouden verkorten. Ook als er wordt gescanned, wordt de config niet opgeslagen. Heel logisch omdat we dan continu de laatst gebruikte frequentie zouden opslaan.

```
bool SaveConfig() {
  for (unsigned int t = 0; t < sizeof(settings); t++)
    EEPROM.write(offsetEEPROM + t, *((char *)&settings + t));
  EEPROM.commit();
  Serial.println("Configuration:saved");
  return true;
}

bool LoadConfig() {
  bool retVal = true;
  if (EEPROM.read(offsetEEPROM + 0) == settings.chkDigit) {
    for (unsigned int t = 0; t < sizeof(settings); t++)
      *((char *)&settings + t) = EEPROM.read(offsetEEPROM + t);
  } else retVal = false;
  Serial.println("Configuration:" + retVal ? "Loaded" : "Not loaded");
  return retVal;
}
```

In LoadConfig zit een controle ingebouwd: Het eerste byte van settings bevat een controlebyte. Als deze niet overeenkomt met de initiële settings, zoals in config.h, wordt de config niet geladen maar worden de settings uit config.h in de EEPROM opgeslagen. Als je dus terug wilt naar de fabrieksinstellingen of als je wijzigingen hebt gemaakt aan de settings stuct, wijzig dan ook het chkDigit in settings. Dit mag elk character zijn, als het maar afwijkt van het vorige character. Zo dwing je het programma terug naar de settings uit de config.h. Doe je dat niet, dan kunnen er, door het lengteverschil van settings, interessante verrassingen in de settings terecht komen.

Kortom, als je aan het programma een variable wilt toevoegen die bewaard moet worden, zet deze dan in de definitie van de settings struct. Zet hem ook in settings in de config.h en verander het chkDigit in settings in config.h. De variabele wordt dan automatisch mee bewaard in de EEPROM.

Het display

De rotatie van het display kan worden ingesteld in de settings. Maar de calibratie van de touch dient dan ook te worden aangepast. Deze kan worden aangepast in de settings.h. Deze regel is hiervoor verantwoordelijk: uint16_t calData[5] = { 378, 3473, 271, 3505, 7 };

Om de juiste cijfers te vinden is er de functie ‘calibrate’ ingebouwd. Deze kan worden geactiveerd met de knop calibrate op pagina 3. Hiervoor dient de transceiver aangesloten te worden op een PC. Druk zo precies mogelijk op de 4 pijlen. De getallen worden via de seriele monitor getoond. De calibratie wordt dus niet echt aangepast, dit dien je zelf te regelen in settings.h. Als het display wordt gebruikt zoals bedoeld, is calibratie niet nodig.

De setup

In de setup gebeurt wat je verwacht dat in een Arduino setup gebeurt: de pindirections worden ingesteld, seriële poorten gestart, display geïnitialiseerd en rotation ingesteld (handig als je het display op z’n kop wilt gebruiken. Denk dan ook aan de alignment van de touch), de instellingen en geheugenkanalen geladen, de DRA ingesteld, WiFi gestart, de APRS en de APRS gateway ingesteld en gestart en de watchdog gestart. Maar ook wordt in de setup de webserver ingesteld. Hierover meer onder ‘webserver’

De loop

Vrijwel alle functionaliteit wordt vanuit de loop geregeld. Dit wil zeggen dat de verschillende functies een variabele zetten die in de loop wordt uitgelezen. Bijbehorende actie wordt uitgevoerd en de variabele wordt gereset. Een van de belangrijke redenen hiervoor is bijvoorbeeld dat een actie van de webserver niet mag worden onderbroken door een redraw van het scherm. Dit eindigt in een crash van de ESP. De afhandeling van de knoppen in de webinterface gebeurt door de variabele ‘Commandbutton’ te zetten. De webserver is dus al klaar voordat de functie is uitgevoerd. De loop regelt de volgende zaken:

- De watchdog wordt gereset.
- De variabele ‘commandButton’ wordt uitgelezen en bijbehorende actie wordt uitgevoerd.
- Het scannen wordt afgehandeld met een interval van 100 mSec.
- Er wordt gekeken of er op het scherm is gedrukt, zo ja dan wordt uitgezocht op welke knop er is gedrukt en wordt bijbehorende functie uitgevoerd.
- De status van de squelch wordt uitgelezen.
- De scan resume wordt afgehandeld.
- De status van de microfoonknoppen wordt uitgelezen.
- Als er een minuut is verstreken, wordt de klok geüpdated.
- De seriële poort wordt gecontroleerd op beschikbare GPS data en de GPS positie wordt geüpdated.
- De status van de PTT wordt gecontroleerd en zonodig wordt de transceiver in TX mode gezet.
- Elke 10 seconden worden de settings opgeslagen, tenminste alleen de gewijzigde bytes worden weggeschreven naar de EEPROM.
- Een APRS baken wordt verstuurd als het zover is.
- De APRS gateway wordt geüpdated als de interval is verstreken.
- Iedere 2 seconden wordt de webpagina gerefreshed.
- Het scherm wordt opnieuw getekend als ‘dirtyScreen’ true is.

De loop heeft het dus best wel wat klusjes, maar een ESP is niet voor een kleintje vervaard en is er binnen een paar milliseconden doorheen. Dus de standaard vertraging van 10 mSec kan geen kwaad. Sterker nog, er zit best wel wat interrupt afhandeling in het programma als gevolg van gebruikte libraries, hierover later meer, stel dat de loop 2 mSec loopt en 10 mSec wacht, dan is de kans dat een interrupt valt in de wachttijd 84%. De kans is dan heel klein dat er

een taak wordt onderbroken. In principe kan het geen kwaad als een taak wordt onderbroken door een interrupt, maar het is wel de basis van veel problemen.

Zoals je kunt zien; in de code van de loop worden geen echte werkzaamheden uitgevoerd. Deze zitten allemaal in losse functies die worden aangeroepen vanuit de loop. Het voordeel hiervan heb ik uitgebreid besproken in het voorgaande verhaal.

Interrupts

Zelf heb ik in het programma geen gebruik gemaakt van interrupts. Maar er zijn wel een paar libraries in gebruik die wel gebruik maken van interrupts. De watchdog is er één van, maar de belangrijkste is die rondom de webserver. Het maken van een connectie en opvragen van een pagina maken allemaal gebruik van interrupts. Zo ook de call-back functie van de APRS. Deze bestaat wel, is bedoeld om APRS packets te ontvangen maar wordt nog niet gebruikt.

Bij de afhandeling van een interrupt moet je altijd in gedachte houden dat het programma op een onvoorspelbaar punt wordt onderbroken en na de afhandeling van het interrupt weer verder gaat waar die gebleven was. Dit is een goede reden om een interruptafhandeling zo kort mogelijk te houden en zo min mogelijk variabelen te wijzigen, zodat er geen situatie kan ontstaan dat een programma op basis van een variabele met een actie bezig is en de variabele onderhand door een interrupt wordt aangepast. Kortom, goed gebruik is dat een interrupt een flag zet en deze flag wordt uitgelezen in de Loop. Dit gaat niet altijd zo makkelijk, maar hierover bij de webserver meer. Als je echt niet wilt dat een routine wordt onderbroken, zet dan aan het begin van de routine de interrupts uit en aan het einde weer aan. En dat is onder andere de reden dat ik, zoals ik in het vorige verhaal schreef, er een hekel aan heb om halverwege uit een routine te breaken. De kans dat je dan vergeet de interrupts aan te zetten (of variabelen te resetten) is groot.

De webserver

Voor de webserver maak ik gebruik van de library ESPAsyncWebServer.h. Hierin zit alles om een complete dynamische webserver te bouwen in een ESP32. Maar allereerst hebben we html pagina's nodig. Deze bouw ik zelf in Visual Code, maar je kunt elke geschikte tool gebruiken om HTML pagina's te maken.

Deze pagina's kun je opslaan in het SPIFFS geheugen van de ESP32, maar het heeft nogal wat voeten in aarde om dit geheugen te vullen. Omdat ik toch voldoende ruimte heb in het programmageheugen van de ESP, heb ik de verschillende pagina's als const char's in webpages.h gestopt. Deze include ik vervolgens in het programma. In webpages.h maak ik gebruik van 'PROGMEM' en 'rawliteral', hetgeen het mogelijk maakt grote complexe blokken data op te slaan in het programmageheugen. In dit geval dus complete html pagina's, er wordt dus geen gebruik gemaakt van het dynamisch ram.

```
const char index_html[] PROGMEM = R"rawliteral(
  <!DOCTYPE HTML>
  <html>
  <head>
    header data....
  </head>
  <body>
    body data....
  </body>
  <script></script>
  </html>
)rawliteral";
```

Om webpagina's te bouwen hoef je ze niet in de ESP te stoppen, maak een html bestand op jouw PC en bekijk het bestand gewoon met een browser. Scheelt een hoop tijd. Denk er ook aan dat webpagina's tegenwoordig allemaal responsive zijn, dan doen ze het ook op een telefoon.

De configuratie van de webserver gebeurt in de setup. Voor elke pagina wordt het volgende stukje code geimplementeerd:

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
  request->send_P(200, "text/html", index_html, Processor);
});
```

Dus als de root van de website wordt benaderd (bijvoorbeeld 192.168.4.1), wordt er een 200 (Dit is OK in HTML) en vervolgens in het formaat ‘tekst/html’ de variabele index_html (dit is dus een complete webpagina) teruggestuurd. Simpel toch?

Voor stylesheets (css files) is een dergelijke functie beschikbaar:

```
server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request) {
  request->send(200, "text/css", css_html);
});
```

In de eerste request->send zit een vierde parameter, ‘Processor’. Deze parameter is de naam van een functie waarin variabelen in de html pagina vervangen worden met de gewenste informatie. Variabelen worden omringd door een % teken. In onderstaand stukje zijn dus %APRSINF%, %GPSINFO% en %BEACONINFO% variabelen.

```
<div class="divinfo">
  <h4 style="text-align:left;color:lightblue"><span id="APRSINFO">%APRSINFO%</span></h4>
  <h4 style="text-align:left;color:orange"><span id="GPSINFO">%GPSINFO%</span></h4>
  <h4 style="text-align:left;color:red"><span id="BEACONINFO">%BEACONINFO%</span></h4>
</div>
```

Dientengevolge is het geen goed idee om het % teken in de html pagina te gebruiken. Ik heb hier flink ruzie mee gehad, ik had een tabel <table width="100%"> dit gaat dus fout. In de css file kun je wel % tekens gebruiken, dus dit is opgelost met <table class="fwidth"> en in de css de volgende class definitie:.fwidth { width: 100%; }.

```
String Process(const String &var) {
  Serial.print("Process: ");
  Serial.println(var);
  if (var == "APRSINFO") {
    FillAPRSInfo();
    return buf;
  }
  if (var == "GPSINFO") {
    FillGPSInfo();
    return buf;
  }
```

In de functie processor wordt de inhoud van de variabele vervangen. (buf is een globale char array). Het vervangen van variabelen gebeurt on the fly, dit kwam goed uit bij het tekenen van de knoppen.
%BUTTONS0% is voldoende om alle knoppen te tekenen in de html pagina. Kijk maar eens in de code hoe ik het heb opgelost.

Voor de verschillende pagina’s zijn deze ‘server.on’ blokken gedefinieerd. Een voorbeeld is de store functie, deze wordt vanuit de settings pagina aangeroepen met <form action="/store" method="get"> hetgeen terecht komt in server.on("/store", HTTP_GET, [](AsyncWebServerRequest *request). Hier een voorbeeld waarbij ik welke functionaliteit uitvoer in de aanroep.

```
server.on("/store", HTTP_GET, [](AsyncWebServerRequest *request) {
  SaveSettings(request);
  SaveConfig();
  SetAPRSParameters();
  request->send_P(200, "text/html", settings_html, Processor);
});
```

Dit is weloverwogen, de functies resulteren niet in een update van het scherm en ze kosten weinig tijd. Na de update ververs ik de pagina door deze opnieuw te sturen. Zou ik een flag zetten en de updates achteraf uitvoeren, dan staat in de webpagina nog steeds de oude informatie en dat willen we niet.

Zo zet ik in server.on("/command", HTTP_GET, [](AsyncWebServerRequest *request) de variabele ‘commandButton’ en ‘dirtyScreen’ op true. In de volgende doorgang van de loop wordt dan het commando uitgevoerd en het volledig scherm ververst. Dit gebeurt dus wel achteraf. Dit werkt allemaal prima als de pagina wordt opgevraagd, andersom willen we ook dat de pagina wordt ververst als er op de transceiver iets wijzigt. Ook hier voorziet de webserver in:

Er wordt bij het connecten van een webclient een eventhandler gestart. Naar deze handler kan data worden gestuurd. Met bijvoorbeeld het commando events.send(buf, "BEACONINFO", millis()); wordt de inhoud van buf gestuurd naar de webclient en komt daar terecht in het aanwezige javascript. Dit script zorgt ervoor dat de inhoud van de inhoud van <span

id="BEACONINFO">%BEACONINFO%
vervangen wordt door de inhoud van buf. Hiervoor is in de webpagina wel javascript noodzakelijk want dit wordt uiteraard client-side afgehandeld.

```
source.addEventListener('BEACONINFO', function(e) {
  console.log("BEACONINFO", e.data);
  document.getElementById("BEACONINFO").innerHTML = e.data;
}, false);
```

Hiervoor is in de webpagina wel javascript noodzakelijk want dit wordt uiteraard client-side afgehandeld.

Libraries

Ik maak gebruik van een aantal libraries:
- Arduino.h: Standaard Arduino library
- SPI.h: Voor communicatie met het display
- TFT_eSPI.h: Display library. Deze staat, correct geconfigureerd, in het project in Github
- WiFi.h: Standaard WiFi library
- WiFiMulti.h: Hiermee kunnen meerdere WiFi netwerken worden gebruikt. Zit in de WiFi library
- EEPROM.h Standaard ESP32 library
- NTP_Time.h: Mooie datum/tijd manipulatie library, zit in de Time library. Deze staat in het project in Github.
- TinyGPS++.h: Voor het parsen van de GPS data. Deze staat in het project in Github.
- LIBAPRS.h: Voor verschillende APRS functies. Deze staat in het project in Github.
- ESPAsyncWebServer.h: Voor de webserver. Standaard ESP32 library
- Esp_task_wdt.h: Watchdog library. Standaard ESP32 library

Afsluitend

Ondanks ruim 2300 regels code, exclusief alle webpagina's, wordt er pas 71% van het programmageheugen gebruikt. Er is dus nog wel wat ruimte om verder te spelen. In de winter ga ik nog wel eens verder met het streamen van audio, het wordt nu te lekker weer. Nog steeds een (vrij zinloze) wens om het aan de praat te krijgen.

Ik ben niet sterk in het documenteren van code, dat probeer ik compenseren door duidelijke veelzeggende namen voor functies en variabelen te gebruiken. Desondanks zal het best even kosten om de code eigen te maken, maar vragen mag altijd. Of we nog iets gaan doen met printen en/of bouwpaketten moet blijken en is vooral afhankelijk van de reacties.

Veel plezier met bouwen!!!

Pim kwam Opa's piephok binnen met een chassis met een aantal buizen erop, die hij van Opa's zolder had gehaald. "Wat is dit?" vroeg hij. Opa keek op en zei: "Oh, dat ding. Dat is een oude super ontvanger." Pim keek verbaasd. "Maar als dat ding zo goed was, waarom heeft u hem dan weg gedaan?" vroeg Pim. Opa schoot in de lach. "Met super bedoelde ik niet dat hij zo goed was, maar dat het een superheterodyne ontvanger was", zei Opa. Nu keek Pim nog verbaasder. "Een superheterowat??", zei hij. "Superheterodyne", zei Opa. "Een ontvanger-type. Een superheterodyne radio-ontvanger, kortweg superhet, is een radio-ontvanger waarin het ontvangen signaal wordt gemengd met een door de ontvanger zodanig opgewekt signaal dat een signaal met een vaste verschilfrequentie, de middenfrequentie, ontstaat. Voor dit signaal kunnen vaste filters worden toegepast om een goede selectiviteit te verkrijgen. De term superheterodyne staat voor het Engelse superaudible (dat is ultrasonisch) heterodyne, wat duidt op het ultrasonische karakter van de opgewekte verschilfrequentie. De toepassing van deze vorm van heterodyne werd uitgevonden door Edwin Armstrong in 1918.

In de tijd vóór de toepassing van heterodyne